UNIT III

7. BASIC PROCESSING UNIT

7.1 Some Basic Concepts

Any program that has to be executed is brought into the main memory. To execute a program processor fetches one instruction at a time from the main memory and performs the operation specified. Instructions are fetched from successive memory locations for the execution until a branch or jump instruction is encountered.

To keep track of the next instruction to be fetched from the memory location, address of the location is maintained in a special purpose register called **Program Counter (PC).** After fetching an instruction before completing its execution, contents of PC are updated to point to the next instruction.

Fetched instruction is sent into the **Instruction Register (IR)** for the execution.

Consider for an example were each instruction is 4 bytes and is stored in one memory word. To execute a single instruction, processor has to perform following three steps:

1. Fetch the contents of memory location pointed to by PC (Program Counter). Fetched instruction is loaded on to IR (Instruction Register).

   $$IR \leftarrow [\,[PC]\,]$$

2. Assuming that memory is byte addressable, increment the contents of PC by 4 bytes so that it will point to the next location in the memory.

   $$PC \leftarrow PC + 4$$

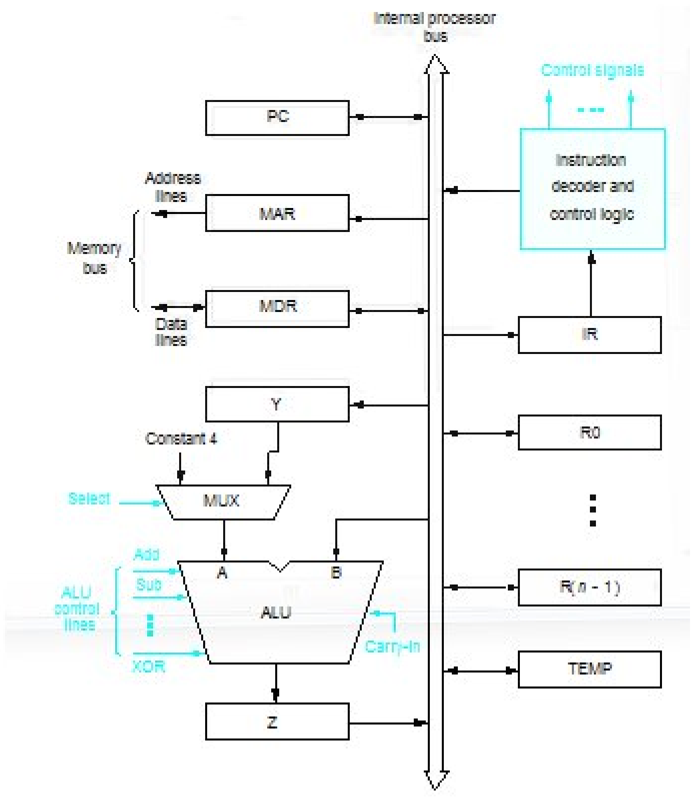
3. The last step is the execution phase which carries out the action specified in the Instruction Register (IR)

Very abstractly, instruction execution is a two phase process were in step 1 and step 2 constitute to **Fetch phase** and step 3 constitute to **Execution phase.**

7.2 Single Bus Organization

The single bus organization exposes the internal organization of the processor with respect to Arithmetic and Logic Unit and different registers that are used for the execution of a instruction.

In the single bus organization, ALU and all registers are interconnected via single common bus called as **Internal Processor Bus.** Hence the name **Single Bus Organization.** Also the organization is capable of implementing only one transfer at a time as all the units share single processor Bus. Below figure shows the architecture of single bus organization with respect to different units.



The organization has different units along the single bus. We will have a closer look into each every unit in terms of its functionality. Processor is connected to the memory unit for communication through Data lines and address lines which are connected to the processor registers MDR and MAR respectively.

1.Memory Data Register (MDR)

Register MDR is used to hold the data that is read from the memory or that needs to be written into the memory.

Register has two inputs and two output ports were one input is connected to processor bus which loads the data from the processor bus and the other input is connected memory unit which loads the data from the external bus of the memory unit that needs to be read.

One of the output ports is connected to processor bus which will send the read data into the processor and other output is connected to external memory data line which will write the data into the memory unit.

2. Memory Address Register (MAR)

Another special purpose register which is used to store the address of the memory location that needs to be accessed. The register has one output and one input line.

Input line is connected to the processor bus which will fetch the required address to be accessed and output is connected to the address line of the external memory bus which will contact the location to be accessed in the memory unit.

Instruction Decoder and Control Logic

This unit is responsible for controlling and co coordinating different operations of the entire unit by issuing control signals to the required unit for their operation. All the units are connected to this unit. Even the control lines of memory bus are connected to instruction decoder and control logic.

It is also responsible for identifying the operations to be performed by the instruction loaded in IR. It also sends the control signals needed to select the registers of the instruction.

Registers

There are n sets of registers and number of these registers varies from processor to processor. Typical register number varies from 16 to 64. Modern processors even come up with 128 registers. They are again classified into two types: **general purpose registers** and **dedicated registers.**

Register Y, Z and Temp

These registers are not specified explicitly in the program instruction but these are the registers used by the processor for temporary storage

Register Y is being connected to processor which is used to store one of the operand fetched for doing the required computation.

Register Z is connected to the output of the ALU which is used to store the result temporarily until it is being transferred to the destination.

Register Temp used for storing the values like Carry out or overflow occurred during the computation.

Multiplexer Mux

There is a 2:1 mux which selects one out of two input values. It selects either the output of register Y which is an operand or a constant value 4 that needs to be provided as a source input A. Constant 4 is used to increment PC to point to the next location in the memory considering that word is of 4 bytes length.

7.2.1 Operations performed on single bus organization

Following are the basic operations performed on any processor which are even done on single bus organization

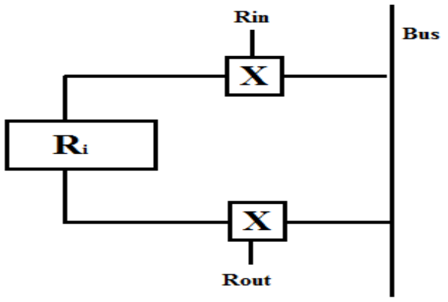
1. Transfer a word of data from one processor register to another processor register or to ALU- **Register Transfers**

2. Perform an ALU operation and store result in a destination location -**ALU operation**

3. Fetch the contents of given memory location and load them into the processor registers -**Load Operation**

4. Store the data from the processor register into a given memory location -**Store operation**

7.2.1.1 Register Transfers



As shown in the above figure, each register uses two control signals, one to place the contents on to the processor bus which is called as **Rout.**

Another control signal is used to load the register with the contents of the processor bus which is called **Rout.**

Input line and output line of any register $R_{in}$and R_{out} are connected to the processor bus via switches and the operations of these switches are controlled by control signals $R_{in}$and R_{out} respectively. When the signal value is 1, the switch is activated or closed or else the switch will be open breaking the connection with processor bus.

When R_{iout}= 1, contents of register $R_i$are placed on the processor bus and when R_{iin}= 1, data on the bus is loaded onto $R_{i..}$When a particular register's $R_{iout}$is 1, other register should not be activated for transferring the output as only one transfer can take place in a single clock cycle.

Example: As an example we will look into the transfer of contents from the register R1 to R4.

Above transfer means R1 is sending its contents as output into the register R4 which will be input for the register R4. Following signals should be activated in sequence for the operation.

1. Enable R_{1out} = 1

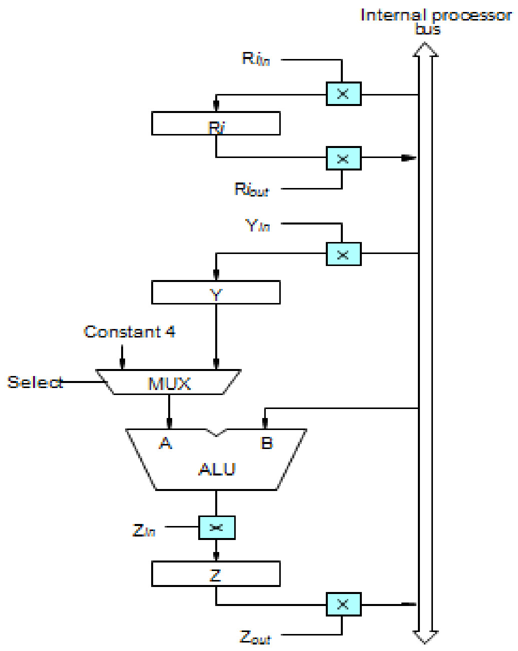
2. Enable R_{4in} = 1

The data transfer takes place according to the clock and only one transfer can take place for a single clock.

7.2.1.2 Performing Arithmetic and Logic Operation

To perform Arithmetic and Logic operations, a processor has a dedicated unit called Arithmetic and Logic Unit were al the required operations are performed based on the control signals sent to them.

Below shown figure gives a very clear picture on the arrangement of basic units that is required for performing any ALU operation.



As according to the single bus architecture shown above, following are the requirements for ALU operation and its corresponding control signals:

→To perform any ALU operation, ALU requires two source operands operand A and operand B as its input

→ One of the operand A is the output of multiplexer which is either the constant value 4 or the contents of the register Y. If the operation is to increment PC, then the operand A is constant value 4 and is selected by the control signal **select4.** If the operand is the contents of some register, then the operand needs to be first stored in the temporary register Y by the control signal **Yin** and the operand is selected by sending the control signal **selecty.**

→ Another source operand, operand B is directly loaded from the internal processor bus connected in common across all the units.

→ Once when the source operands are ready on the input of ALU A and B, required operation is done in the ALU by sending the corresponding control signal indicating the operation and the result is stored inside the temporary register Z through the control signal **Zin.**

→ Finally required result is transferred from the temporary register into the destination loaction.

Summary of Actions:

1. First load the contents of one of the operand into the temporary register Y.
2. Fetch another operand from the register and load it onto the processor bus. When both operands are fetched perform the required ALU operation.
3. Store the computed result in the destination register.

Example: Add R1, R2, R3.

To know how exactly the control signals are generated, let us consider the above instruction as an example. The above instruction adds the contents of two registers R1 and R2 and stores it inside the destination register R3. Before writing the sequence of control signals, let us know the series of actions that needs to be performed for the above instruction.

Actions:

1. Load the contents of R1 into the temporary register Y
2. Load the contents of R2 onto the processor bus and perform the required ALU operation and store it inside the temporary register Z.
3. Finally transfer the computed result from Z into the required destination loaction.

Following are the sequence of signals according to above mentioned actions:

1. $R1_{out}$, Y_{in}	(Load contents of R1 into Y)
2. $R2_{out}$, SelectY, Add, Z_{in}	(Perform ALU operation)
3. Z_{out}, $R3_{in}$	(Store result in destination location)

Note : Each of the above step mentioned takes at least one processor clock cycle to complete their operation

7.2.1.3 Fetching a word from the memory location

Fetching a word from a memory location means it can be both data operands as well as the program instructions read from the memory location.

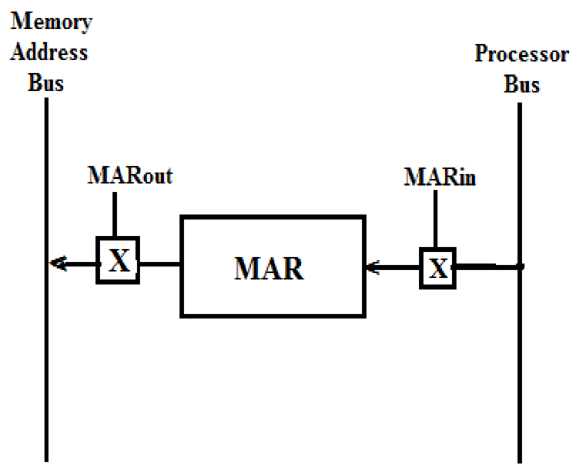
To fetch any word, processor has to specify the address of a memory location along with the read request into the memory unit.

To fetch a word from the memory location, following are the actions that has to be prformed in sequence:

1. Load the register MAR with the address of the required location to be accessed along with the read control signal.
2. Requested data is then loaded into MDR.
3. From MDR it is sent to the processor through the processor Bus.

To perform the above actions in sequence, MAR and MDR units are involved in the operation. We will look into the required control signals to control and coordinate their actions.

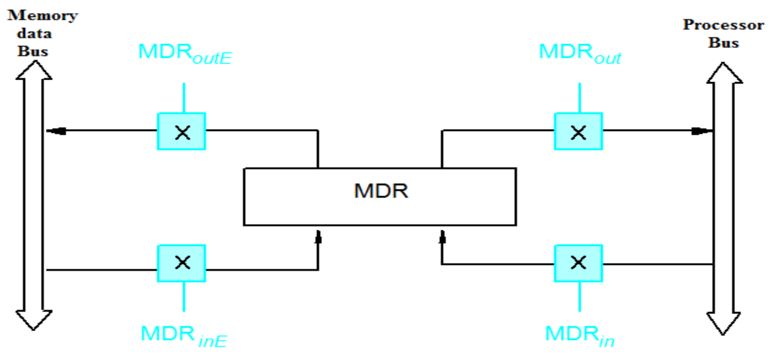
MAR Unit:



MAR unit has one input line and one output line, input line is connected form processor bus and the output line is connected into the address bus of the memory unit.

To trigger the input and output operation of the MAR, it has two switches connected to its line which are controlled by the control signals **MARin** for the input and **MARout**for the output line. MARout is always active and MARin is activated by providing the signal at its switch.

MDR Unit:



→ MDR unit has two input and two output lines. One of the input lines is connected to processor bus which loads the data from the processor register which needs to be written into the memory unit. This line is controlled through a switch by a control signal **MDRin.**

→ Another input line is connected to external memory data bus which loads the data from the memory unit that needs to be sent into the processor. This line is controlled through a switch by a control signal **MDR_{inE}.**

→One of the output line is connected to a processor bus which is used to load the required data from the memory unit into the processor register. This line is controlled through a switch by a control signal **MDR_{out}.**

→Second output line is connected to memory data bus which writes the loaded data from the processor into the memory unit. This line is controlled through a switch by a control signal **MDR_{outE}.**

One of the important point to be noted here is that all the internal processor operations completes the operation in one clock cycle but when the operation is related to external device such as memory unit, it will take more than one clock cycle as there will be an added delay when there is communication with the device which is connected on the external circuit. So it is very much important for the processor to wait more than one clock cycle till the external unit completes its operation.

To know if the operation is completed or not, there is a dedicated control signal attached with this unit which is **MFC (Memory Function Completed)** signal which signals the completion of the operation done by the memory unit. Processor can then proceed with the further operation on receiving this signal.

Following are the complete summary of actions:

1. Load MAR with required address along with the issue of Read signal.

2. Load the fetched data from the memory bus into MDR and wait for MFC signal.
3. On receiving MFC, transfer the loaded data into the required processor register.

Example 1:

Move (R1), R2

Consider the above example and we will know how exactly the sequence of control signals are generated accordingly.

→ Above instruction reads the data from the memory address pointed to by the contents of register R1 and loads into the destination register R2.

→ Following are the sequence of actions performed for completing the operation:

1. Load MAR with the required address to be accessed along with the Read signal.
2. Load MDR with the data transferred from the memory bus and wait for MFC signal.
3. Transfer the contents of MDR into required processor register.

Following are the sequence of signals according to above mentioned actions:

1. $R1_{out}$, MAR_{in}, Read	(Send the address to be accessed into MAR)
2. MDR_{inE}, WMFC	(Transfer the data into MDR and wait for MFC)
3. MDR_{out}, $R2_{in}$	(Transfer the contents from MDR into R2)

Note: In the second step, MDR_{inE} can be omitted as it will be active until WMFC.

Example 2: Add (R1), R2

Following are the actions to perform the above operation:

1. Fetch the operand from the memory location pointed to by the contents of R1

2. Fetch another operand from the register and load onto processor bus.

3. Perform the addition

4. Store the result in destination register R2

1. $R1_{out}$, MAR_{in}, Read
2. $R2_{out,}$ $Y_{in,}$ WMFC
3. MDR_{out}, Selecty, Add, Z_{in}
4. $Z_{out,}$ $R2_{in}$

7.2.1.4 Storing a word in a memory

Storing a word of data into the memory location includes all the units that are involved in reading a word from the memory location. The same set of control signals are again used but for the different set of actions according to the requirement. Finally to be very precise it is the communication between processor and memory unit

Let us first highlight the general set of actions that takes place to store a word into the memory location:

1. Load MAR with required address to be accessed in the memory location for writing the data value.
2. Next Load MDR with the data from the processor registers that needs to be written into the memory unit along with the Write signal.
3. Wait for MFC signal to proceed with the further activity.

Let us illustrate the above mentioned actions taking an example of the instruction:

Move R2, (R1)

The above instruction stores the contents of register R2 into the memory location address pointed to by the contents of register R1. Following are the sequence of actions that are performed:

1. Load MAR with the contents of register R1.
2. Load MDR with the contents of register R2 and issue write signal.
3. Wait for MFC signal.

Following are the sequence of signals according to above mentioned actions:

1. $R1_{out}$, MAR_{in}
2. $R2_{out}$, MDR_{in}, Write
3. MDR_{outE}, WMFC

Note: In the third step, MDR_{outE} can be omitted as it will be active until WMFC.

Example 2: Add R2, (R1)

Following are the set of actions performed to complete the operation:

1. Fetch operand B from the memory location pointed to by the register R1.
2. Fetch operand A from the register R2 and load on to the register Y.
3. Perform the addition operation on two operands and store the result in register Z.
4. Move the result into the memory location pointed to by the MAR.

Following are the sequence of control signals accordingly:

1. $R1_{out,}MAR_{in}$, Read
2. $R2_{out,}$ $Y_{in,}$ WMFC
3. $MDR_{out,}$ SelectY, Add, Z_{in}
4. $Z_{out,}$ $MDR_{in,}$ Write
5. WMFC

7.3 Execution of a Complete Instruction

We know that execution of complete instruction is abstractly classified into two phase process:

1. **Fetch**
2. **Execute**

Fetch phase fetches the required instruction from the memory location into the instruction register for the purpose of later execution

Execute phase takes the responsibility of identifying the operation involved performs the required task.

In this section, let us put together the sequence of all elementary actions that is required for generating the control signals to fetch and execute a single instruction completely.

To illustrate the process, let us consider the example given below:

Add (R3), R1

Meaning: Fetch the contents from the memory location pointed to by the register R3 and add it with the contents of R1 and store the result back in R1

Execution of this complete instruction has following steps:

1. Fetch the instruction from the memory.
2. Fetch operand from the memory location an d load it into Y
3. Fetch operand from R1 and perform the addition.
4. Store the result into the register R1

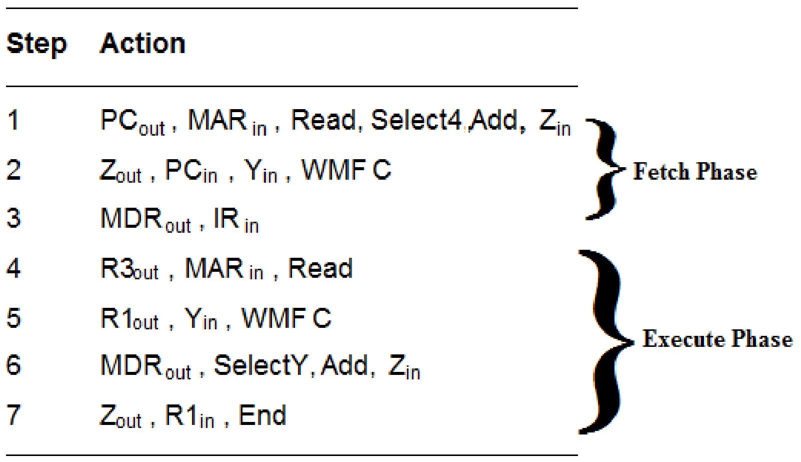
The above mentioned steps include all the required actions which explain both fetch and execute phase. But fetch phase itself has a few more steps which is not mentioned in the above set of actions and is mentioned very abstractly in a single step.

Following are the actions required for fetching any instruction from the memory:

1. Send the address present in PC to MAR along with the Read signal and increment PC by adding it with the constant value 4
2. Update PC with the new value and wait until MDR finishes loading the instruction from the memory.
3. Send the loaded instruction from MDR into the Instruction Register (IR).

Note: Above mentioned steps are common irrespective of any instruction were the execution always begins the fetch phase. It is only the execution phase varies according to the type of the operation which we have already seen.

Following are the sequence of control signals generated for the instruction Add (R3), R1:



Step	Action	
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}	
2	Z_{out}, PC_{in}, Y_{in}, WMF C	Fetch Phase
3	MDR_{out}, IR_{in}	
4	$R3_{out}$, MAR_{in}, Read	
5	$R1_{out}$, Y_{in}, WMF C	Execute Phase
6	MDR_{out}, SelectY, Add, Z_{in}	
7	Z_{out}, $R1_{in}$, End	

Note: Similarly try to write the sequence for all the instructions given as an example

7.4 Branch Instructions

So far we have seen the generation of the control signals with respect to different kinds of sequential instructions. But we haven't had a discussion if in case it is a branch instruction. In this section we will look into the technique of generating control signals for the branch instructions:

There are two types of branch instructions:

1. **Unconditional Branch instructions**
2. **Conditional Branch instructions.**

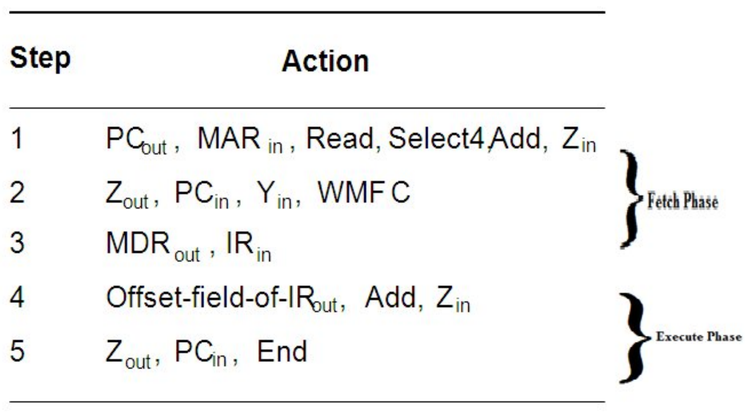
7.4.1 Unconditional Branch instructions

Unconditional branches are the instructions that change the contents of the PC to the Branch Target Address specified in the instruction without any condition satisfaction.

When this instruction is executed, it just replaces the contents of the PC with the new value of BTA (Branch Target Address). BTA are usually a displacement value X which is to be added to the contents of PC specifying the displacement with respect to PC having the form X(PC).

Processing of this type of instruction starts as usual with the fetch phase and ends with the execution phase which changes the contents of PC. The execution phase decodes the offset value required and updates PC by adding the offset value to the PC.

Following are the sequence of control signals generated for executing unconditional branch instruction. First three steps are common representing the fetch phase. The last two steps represent the execute phase in which step 4 produces the exact Branch target address to be loaded into PC and step 5 updates PC with calculated BTA.



Step	Action	
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}	
2	Z_{out}, PC_{in}, Y_{in}, WMFC	Fetch Phase
3	MDR_{out}, IR_{in}	
4	Offset-field-of-IR_{out}, Add, Z_{in}	Execute Phase
5	Z_{out}, PC_{in}, End	

7.4.2 Conditional Branch Instruction

Conditional branches are the instructions that change the contents of the PC to the Branch Target Address specified in the instruction if the condition is satisfied.

When this instruction is executed, it will check if the condition is satisfied by referring into the values of condition codes. If condition is satisfied, it just replaces the contents of the PC with the new value of BTA (Branch Target Address). If condition is not satisfied it will continue with the fetching new instruction in sequence ending the execution part.

Processing of this type of instruction again starts as usual with the fetch phase and ends with the execution phase which changes the contents of PC only if the condition is satisfied or else terminates execution and loads the new instruction by starting the fetch phase. To check the condition, it checks the condition code values.

As an example lets us consider the instruction Branch < 0, this instruction updates PC value only if the condition code N is set to 1. If N value is set to 0, it will fetch the new instruction in sequence.

Following are the sequence of control signals generated for executing conditional branch instruction Branch < 0. First three steps are common representing the fetch phase. The last two steps represent the execute phase in which step 4 produces the exact Branch target address to be loaded into PC and if N=0, it will start from the step 1 which loads new instruction or else goes to step 5 which updates PC with calculated BTA.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Add, Z_{In}, if N=0 then End
5	Z_{out}, PC_{in}, End

7.5 Multiple Bus Organization

There was a major disadvantage in the single bus organization were in organization allows only one transfer at a time as all the units connected in a single bus organization shared a single internal processor bus to do transfer from one unit to another unit. Because of this it even affected the performance of processor in terms of increasing number of clock cycles for the operation.

To overcome this disadvantage multiple bus organization aims at enabling the several transfer in parallel using multiple lines for the data transfer connected across various units of the architecture.

One of the multiple bus organizations which we are going to study is the three bus structure.

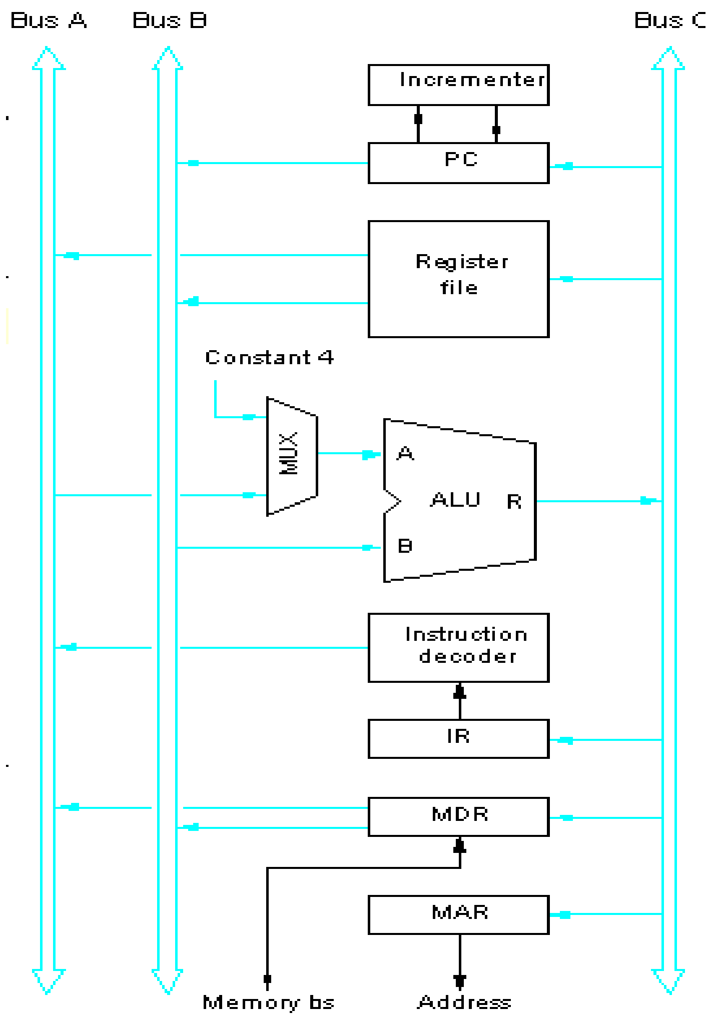
7.5.1 Three Bus Organization

As the name indicates, the architecture has three different buses to which different units are connected.

1. Bus A
2. Bus B
3. Bus C

Bus A and Bus B is used to transfer source operand A and B into ALU inputs A and B respectively. Bus C is used to transfer the result into the required destination location.

Apart from that there are different units connected across the architecture which are almost same apart from very small changes in its functionality. We will look into the units and their functionality one by one. Below figure shows the three bus organization:



→ **Register File:** In this architecture, all registers are combined together as register file. A register file is having three ports in which one of them is input port and two are output ports.

One output port is connected to Bus A and another output port is connected to Bus B. This allows contents of two registers to be transferred simultaneously. Control signals for doing these operations are R_{ioutA} and R_{ioutB} respectively. Were i is the register number

One of the input is connected to Bus C which fetches the data to be stored inside the register. Control signal to do this operation is $R_{iin.}$

→ **Incrementer Unit:** This is a new unit introduced in this architecture which takes care of automatically incrementing PC by 4. Even then the constant 4 is used which is used to increment other address like memory unit.

Control signal to invoke this unit is Incpc.

→ **Program Counter (PC):** This unit has one input and one output. Input connected to Bus C and output to Bus B. Control signals are PC_{in} and PC_{out} respectively.

→ **ALU:** This unit has two input for fetching source operand A and B respectively. Operand A is fetched from the MUX which has constant 4 as one input and the other one connected to Bus A. Operand B is fetched directly from Bus B. There is one output R connected to Bus C.

→ **Instruction Register (IR):** Has one input port connected to Bus C.

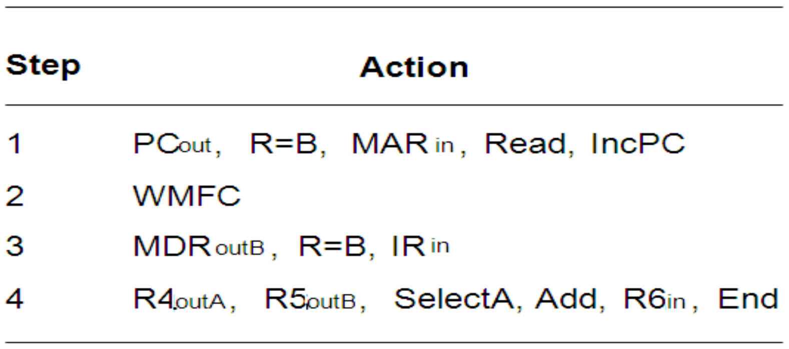
→ **Memory Data Register (MDR):** It has two outputs and one input. Two outputs are connected to BusA and BusB respectively which is controlled by the signals MDR_{outA} and $MDR_{outB.}$ One input is connected to Bus C which is controlled by signal $MDR_{in.}$

→ **Memory Address Register (MAR):** It has one input connected Bus C which uses the control signal $MAR_{in.}$

Note that all the units have their input line connected to Bus C and ALU is the only unit which has its output connected to Bus C. So to transfer contents from any two units, transfer has to be done via ALU output port through Bus A or Bus B. ALU can select any one input port by the signal R=A or R=B which selects either input on port A or port B respectively.

We will consider the example Add R4, R5, R6 and check how the control signals are generated according to the 3 bus organization. Again there are two phases fetch and execute. Fetch is common for all the instructions. Below shown is the generation of signals:

This instruction fetches operands from register R4 and R5 and stores in the destination register R6.



Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

7.6 Methods for generating control signals

So far we have seen how exactly control signals control and co ordinate the complete execution of the instructions. We have also seen the different control signals that are generated by the control unit to do different operations.

But there has to be some means (method) for generating the control signals through the control unit. Under this concept we will look into the insight of the control unit and let us look into the method of generating them.

To generate a control signals through control unit, there are two methods:

1. Hardwired Control
2. Microprogrammed Control

We will look into each one of them in detail.

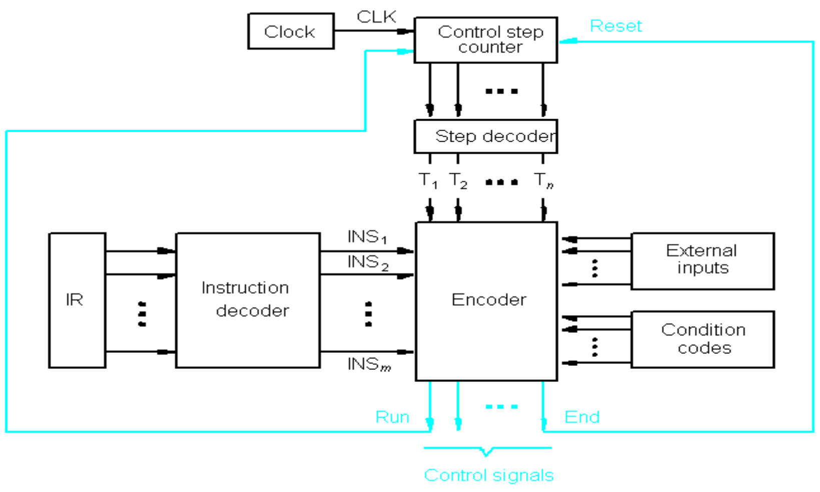
7.6.1 Hardwired Control

As the name indicates, the unit to generate control signals is a hardware based unit which is a fixed unit consisting of wiring of logical gates through which the required control signals are generated in time.

To generate any control signal, unit requires four basic things:

1. Step Number
2. Contents of Instruction Register
3. Contents of condition Codes (Branch instructions)
4. External signals such as MFC etc (Memory Operation)

Considering the above basic requirements, hardwired functional units are implemented. Below shown is the complete organization of a Hardwired unit.



Following are the units involved in the operation and their functionalities are:

1. **Control Step Counter:** This unit keeps track of the step numbers for generating the signals and for every clock cycle it increments the step number whenever Run signal is set to 1.
2. **Step Decoder:** This unit provides a separate signal line for each step or time slot in the control sequence.
3. **Instruction Register (IR):** We know that this is a dedicated register that is meant for holding fetched instruction for the later part of execution which is directly connected to the decoding unit.
4. **Instruction Decoder:** This unit takes in the instruction as the input, decodes it, and generates a separate signal line for each of the instruction. Here one of the output line from Ins1,…….Insn is set to 1and all other lines are set to 0.
5. **Encoder Block:** This is the block that is responsible for generating the required control signals through logical gate connection. This block has four input lines, one of them is connected to the output of the decoder, another to step number, the third one to external input and the final one to condition codes. It combines the various input in this block through the logical gates to generate a required control signals.
6. **Clock:** Clock is used to generate timing signals required for the actions to be performed by a particular step. All the step takes at least one clock cycle for the operation.

Apart form that there are two control signals **Run** and **End.** Run=1, increments the step counter for every clock cycle. and Run=0 halts the step counter. This is done when processor is waiting for some external operations.

End=1 resets the step counter and starts from step number 1. This is done when fetching the new instruction.

Next we will see how the control signals are generated through the logical gating which uses logical expressions. For that we will consider the following two control signal sets. Let the first one be Add and second one be named as Branch.

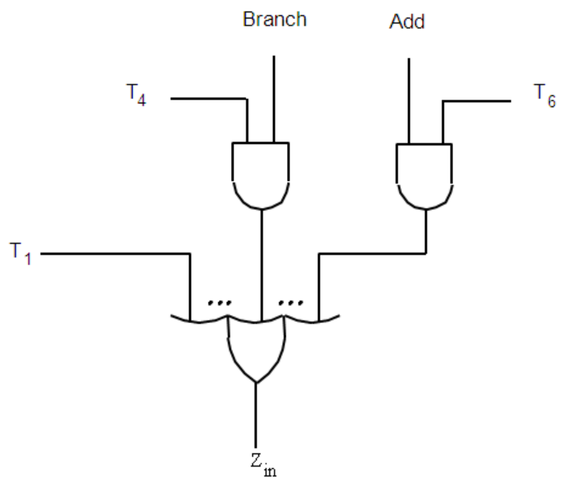
	Add		Branch
1	PC_{out} , MAR_{in} , Read, Select4,Add, Z_{in}	1	PC_{out} , MAR_{in} , Read, Select4,Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMF C	2	Z_{out} , PC_{in} , Y_{in} , WMF C
3	MDR_{out} , IR_{in}	3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read	4	Offset-field-of-IR_{out}, Add, Z_{in}
5	$R1_{out}$, Y_{in} , WMF C	5	Z_{out} , PC_{in} , End
6	MDR_{out} , SelectY, Add, Z_{in}		
7	Z_{out} , $R1_{in}$, End		

We will first write the logical expression for generating the control signal Z_{in} with respect to the above mentioned two instructions. Next based on that we will look into the gating arrangement that is being done.

Note that first three steps of fetch are common irrespective of the type of instruction that is being executed. Logical expression is given by:

$$Z_{in} = T_1 + T_6 \bullet ADD + T_4 \bullet BR$$

Corresponding gating arrangement is:



Note : Similarly try it for other signals too. In case of any doubts feel free to get it clarified.

7.6.2 Microprogrommed Control

So far we have seen the technique of generating control signal through the hardware based unit implemented through connecting the logic gates.

In this section we will discuss the technique of generating control signals through software technique called as **Microprogrammed Control.**

As the name indicates, in this technique control signals are generated by program similar to machine language programs. Let us see how microprogram signals are generated for the following sequence:

1	PC_{out} , MAR_{in} , Read, Select4,Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMF C
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMF C
6	MDR_{out} , SelectY, Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

Table below shows the microprogram instructions for generating the control signals. All the required control signals are generated in a sequence for a particular step and which ever signal is used, the value of that is set to 1 or else to 0.

Micro - instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1

Following are the terms associated with generating microprogram instructions:

1. **Control word:** It is a word whose individual bits represent the control signal which is a combination of 0's and 1's. Each step is represented by a control word. These are also called as **Microinstruction.**
2. **Microroutine:** A sequence of control words corresponding to sequence is called as **microroutine.**
3. **Control Store:** Microroutines for all the instructions are stored in a special memory unit called as **control store.** Required microroutine is fetched from the particular address of the control store.

Control unit generates the control signal by sequentially reading control words corresponding to a microroutine of a particular instruction.

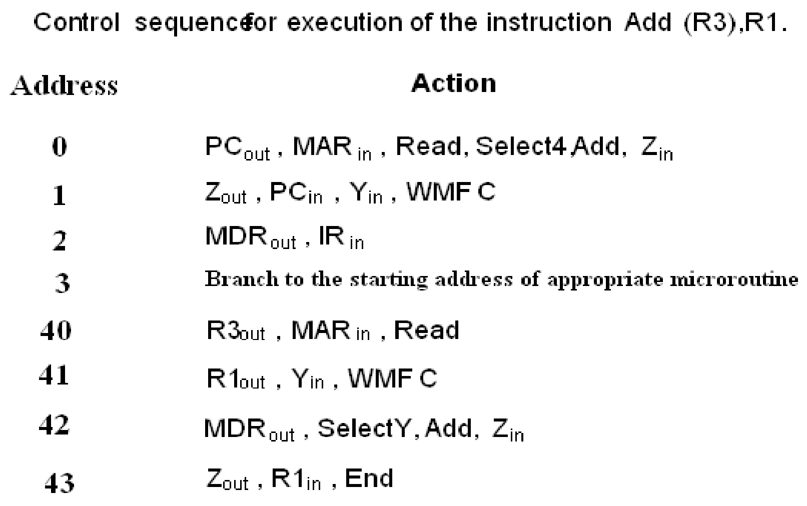
We know that any instruction execution has two phases:

1. Fetch
2. Execute

1. When it comes to the fetch phase it remains same for all the instructions irrespective of the type of instruction.

2. Once after fetching the instruction, execution of an instruction has to load the control signals of a particular microroutine from a particular address in the control store. This process is called as **Branching to the microroutine.**

To illustrate this process let us consider the example of generating control signals for instructions Add (R3), R1 through microprogram which is shown as follows:



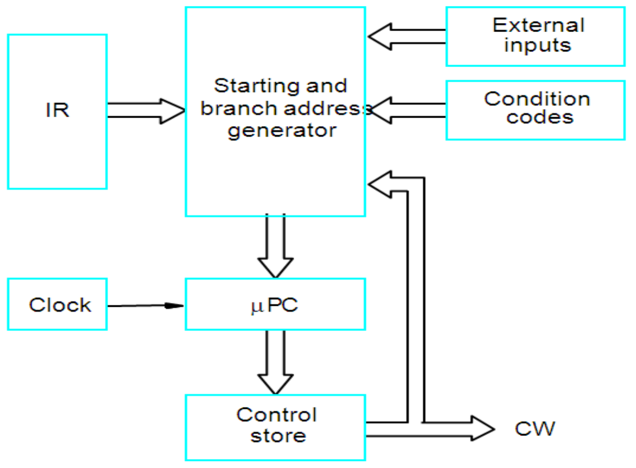
Control sequence for execution of the instruction Add (R3),R1.

Address	Action
0	PC_{out} , MAR_{in} , Read, Select4 Add, Z_{in}
1	Z_{out} , PC_{in} , Y_{in} , WMF C
2	MDR_{out} , IR_{in}
3	Branch to the starting address of appropriate microroutine
40	$R3_{out}$, MAR_{in} , Read
41	$R1_{out}$, Y_{in} , WMF C
42	MDR_{out} , SelectY, Add, Z_{in}
43	Z_{out} , $R1_{in}$, End

Operations:

1. Any instruction that starts its execution starts with the fetch phase which starts from the address 0 in the control store.
2. Control words from address 0 to address 3 are generated for all the instructions irrespective of type as it is a fetch phase.
3. In the address 3 there is as a control word finding the type of the instruction and loads the address of the microroutine corresponding to that instruction. The process is called branching.
4. Then the control words from the control store are generated in sequence till it reaches end.

To implement the above mentioned operations, micro programmed unit is organized as follows:



We will know the functionality of the following units:

1. Starting address and branch address generator: This unit is responsible for generating the starting address which starts the fetch operation and also responsible for generating the required microroutine address which branches the control to the required address in the control store for generating sequence of control words corresponding to the required instruction.

2. µPC: This unit is responsible for reading the control word sequentially in an incrementing order which increments the address of control word sequentially for every clock cycle. The initial and branching address is taken from Starting address and branch address generator.

Working:

1. First the instruction is fetched by loading the address zero into the starting address and branch address generator which reads the fetch microprogram from the control store.
2. Then the µpc increments the address sequentially up to address 3.
3. In step 4 starting address and branch address generator will generate the address of the microroutine corresponding to a particular instruction. Then their control words are read sequentially from the control store till the process is completed.
4. On reaching the end control signal, starting address and branch address generator is loaded with the address 0 corresponding to fetch microroutine which fetches new instruction.